Разработка blockchain на python

Оглавление

1.1. Технология блокчейн

Блокчейн – распределённый, децентрализованный и часто общедоступный реестр с растущими блоками (аналогия связного списка), связанными друг с другом посредством криптографических хешей.

Первый блок в подобной цепи именуется Генезис-блоком, является уникальным, т.к. генерируется разработчиком сети и не имеет предыдущего. В дополнение к данным транзакции в блоке также находится хеш предыдущего блока, временные данные.

Под временем блока понимают среднее время необходимое сети для генерации последующего блока в цепи, после чего цепь становится проверяемой. Чаще всего это время в криптовалютах это время прохождение транзакции ~ 10 мин для Bitcoin.

Часто может быть представлен в виде дерева Меркла. Так как каждый блок содержит связь с предыдущим, блоки могут быть объединены в цепочку, что обеспечивает неизменность передаваемых данных, т.к. изменение данных даже в одном блоке приведёт к изменению всей цепочки.

Чаще всего блокчейн управляется через P2P одноранговую сеть как публичный распределённый реестр. Каждый узел в такой цепи должен придерживаться определённого консенсусного набора правил.

Хотя блоки и не могут быть изменены задним числом, существует возможность изменения данных в блоках путём создания fork (вилка). Между тем блокчейн принято считать безопасной вычислительной системой.

Soft fork и hard fork. Мягкая вилка или просто вилка – это изменение в протоколе блокчейна, когда модифицируемый или добавляемый функционал не вносит значительных изменений в структуру и работу. При этом новые блоки, перестают воспринимать узлы

старой версии, определяя их как недействительные. Если же старые блоки придерживаются консенсуса, то они могут быть приняты. Так можно говорить о обратной совместимости мягкой вилки. При внесении таких изменений достаточно большинства для получения консенсуса, при том, что каждый пользователь не обязан принимать новые правила и запускать обновлённую версию кода. Зачастую данный механизм использовался многими блокчейн технологиями для внесения обновлений на лету, исправления ошибок и расширения сетевых возможностей. Наиболее известные примеры для bitcoin – SegWit и Taproot.

В определённые моменты может возникнуть hardfolk (жёсткая вилка), приводящая к невозможности обратно совместимым ввиду изменения протокола сети и требует обновления ПО со стороны пользователя. Таких ситуаций принято максимально избегать, так как они приводят к техническим сложностях и финансовым потерям.

Важнейшими преимуществами блокчейна при использовании в цифровых активах являются отсутствие бесконечной воспроизводимости и возможность недорогой и быстрой проверки со стороны пользователя

Под завершённостью понимают уровень уверенности, при котором корректно сформированный последний блок, добавленный в сеть, не будет доработан в будущем. Ни один из двух наиболее распространённых консенсусов не обеспечивает этого, они полагаются на вероятностную завершённость, когда вероятность изменения блока тем меньше, чем большее количество блоков было создано.

Под консенсусом в блокчейн понимают одно из двух правил:

- Proof of work (подтверждение работы), изначально подход использовавшийся для защиты от DoS атак, при котором одна сторона в виде Проверяющего доказывает другой стороне Верификатору факт затрачивания определённого количества вычислительных усилий, что требует высоких затрат энергии и вычислительных мощностей. PoW требует от участников сети выполнения неких сложных вычислительных задач, что доказывает затрату N-го количества ресурсов со стороны конкретного пользователя. Первый участник цепи, который совершил вычисление, может добавить новый блок в цепь. Сложность задач регулируется динамически и позволяет сохранять постоянным среднее время решения задач.
- Proof of stake (подтверждение доли) требует, чтобы валидаторы имели некоторое количество токенов блокчейна необходимых потенциальным

злоумышленникам приобрели, зачастую им необходимо иметь большую часть токенов в блокчейне для проведения атаки. Наиболее энергоэффективный метод. Вероятность добавления нового блока в цепь пропорциональна имеющемуся у пользователя количеству добавленных ранее блоков.

- Leased Proof of Stake (арендованное подтверждение доли), предусматривает возможность сдачи в аренду части своей доли узлу или узлам сети, для решения проблемы привлечения новых пользователей, присущей предыдущему типу консенсуса. На ряду с арендой существует и подход делегирования Delegated Proof of Stake, при котором пользователь делегирует возможность добавления блоков в выбранный узел, но при этом пользователи в нём не получат вознаграждения при добавлении нового блока.
- Proof of Importance (подтверждение важности), подразумевает механизм используемый в LPoS с учётом не только количества добавленных блоков, но и сетевой активности пользователя.

Часто используется несколько консенсусов одновременно в одной сети на разных этапах её развития. PoW на начальном этапе расширения сети и накопления блоков и позже PoS для поддержания сети.

1.2. Задание

Необходимо создать простейший blockchain на python, имитирующий работу цепочки блоков, связанных между собой с помощью хеширования и ссылки на предыдущий блок. Блоки содержат внутри себя некоторые данные, представленные в текстовом виде, время их создания, хеш предыдущего блока, работающий по консенсусу proof of work. Для цепи необходимо реализовать проверку её валидности путём сравнения хешей рассчитываемых блоков. Возможность указания временной сложности добычи блока и сложности хеширования в формате “00000000”.

1.3. Создание blockchain на python

Перед началом работы следует подключить библиотеку hashlib, поддерживающую OpenSSL, позволяющую работать с различными криптографическими функциями, а также библиотеку time. Под блокчейном примем некую систему, где необходимо выполнить некую работу для добавления нового блока в цепь.

Для создания блокчейна необходимо реализовать классы: block, blockchain.

1.1.1. Класс Block. Описывает отдельный блок в цепи, содержит такие поля как:

- временная метка (timestamp) – время создания блока, используемое для расчёта хеша в дальнейшем
- данные (data) – какие-либо данные передаваемые в блоке при транзакции
- хеш предыдущего блока, на который ссылается текущий, обозначаем за Null для первого блока, т.к. предыдущего у него нет
- Уникальное случайное число (nonce) – случайное или псевдослучайное уникальное число, которое используется только единожды. Используется как переменная в процессе добычи блоков. Необходимо найти подходящее число, генерируемое хешом, соответствующее определённым условиям, в таком случае создаётся блок. Сложность принято задавать количеством нулей “000000”.
- Хеш текущего блока, получаемый с помощью метода расчёта хеша

1.1.2. Метод getHash(), возвращающий хеш, использует SHA256, рассчитывается на основании предыдущего хеша, временной метки, данных, nonce. Для этого их необходимо приводить к необходимой кодировке encode('utf-8'), а также приводить к строке STR()

Для обновления хеша используется hash.update от перечисленного выше

Далее необходимо перевести полученное к побитовому виду hexdigest()

1.1.3. Метод mine(), вычисляющий хеш до тех пор, пока не будет найден нужный по заданной сложности. С каждым проходом инкриминируем nonce и обновляем значение hash. Сложность вычисляется как количество нулей ‘0’*difficulty

1.1.4. Класс Blockchain. Содержит сложность, временную сложность вычисления блока в мс, саму цепочку блоков. В нём необходимо реализовать следующие методы:

- Получение последнего блока
- Добавление нового блока в цепочку
- Проверка цепи (проверяет хеши блоков и предыдущих блоков, проходясь поочередно по цепи, i начиная с 1, т.к. первый блок задаётся пользователем, а не рассчитывается по предыдущим

На этом минимальное необходимые для реализации моменты заканчиваются, использование nonce и сложности позволяет провести проверку proof of work для цепи.

Остается создать первый блок, рассчитать последующие и записать в файл результат работы программы в следующем формате для текущего блока и предыдущего:

Data:
Timestamp:
Nonce:
Hash:
PrevHash:

Для реализации задания опционально можно использовать http запросы, flasc и прочие сетевые фреймворки.